GTU

**DEPARTMENT OF COMPUTER ENGINEERING**

**CSE 463 – Spring 2022**

**HOMEWORK 3  
REPORT**

SÜLEYMAN GÖLBOL 1801042656

1. **PROBLEM SOLUTION APPROACH AND ALGORITHMS**

My problem that I have encountered in application was about image selecting.

I didn’t want to use global variables to store the value of input path. So I made inside of my ImageSelecting module as a class. In that class, I needed to hold values of some variables inside constructor . To initialize them, I holded them as string list.

Text

Description automatically generated

But after that, problem was checking if both images are selected and it wasn’t easy. Because if no images are selected, images[i] values were None. But when one of them selected both of them changes. So put another if condition to check if(self.images[0] == “”). But this had an other problem. “FutureWarning: elementwise comparison failed; returning scalar, but in the future will perform elementwise comparison”. The warning was because of problem between python and numpy arrays. Python thinks Scalar or a Numpy tkinks ndarray. So, instead I checked the lengths.

A screenshot of a computer

Description automatically generated with medium confidence

Other thing was rectification but because of example images are already are rectified, I didn’t need to rectify them again.

I have 3 types of features based matching algorithm which are inverted and combined.

For sift features, I used “SIFT\_create()” method from OpenCV.   
After this sift variable created, I converted images to grayscale for simplicity then used detectAndCompute(image,None) to get the keypoints and descriptors.

Then I’ve created a numpy array called disparity\_map which is full of zeros with size (height, width) of image (left or right, they have same size)

Then I matched with cv.BFMatcher().match(descriptor1, descriptor2) which is brute force match. I used knnMatch to get the matches and then to get the best matches I checked for the good matches.

Text

Description automatically generated

**Calculating disparity (algorithm)**

Text

Description automatically generated

While I am calculating disparity by myself; I got the coordinates of match points. So, in a for loop, I used queryIdx to get the keypoint info from left image and I used trainIdx to get the keypoint info from the right image.

After getting keyPoint index info, I used “.pt” to get the coordinates.

For example keyPoints[match.queryIdx].pt returns 2 numbers (x\_img1, y\_

img2). These are the coordinates of the points.

Then for each point in left and right images; I found the disparity by using subtraction and I used abs() to get the absolute value of it. Because disparity is xCoord2-xCoord1. I multiplied by 8 because, provided images have small corresponding pixel differences. To see them clearer on the image, I used this.

Feature based methods that I used while I was calculating disparity was sift detector, classic brute force matcher, norm hamming matcher, flann feature matcher, gradient of gaussian edge detector, orb detector.

Because of these are feature based, my disparity images are sparse. (not dense)

**General Algorithm of Disparity Finding**

Create Sift/Orb/(fast+brief) object

Turn images into grayscale

if (edgeDetector selected)

turn grayscale images to edge images with Canny

Detect keypoints using fast object created above

Detect descriptors using brief object and keypoints

else

Detect keypoints and descriptors using object

Create matcher object (L2/NORM HAMMING/FLANN)

Call match() / knnMatch() using descriptors on arguments

Find good matches by checking distances of >%70 difference

Call drawMatches() to see the corresponding matches

Create empty disparity map with image height x width

For every match in good Matches

Get point coordinates from keyPoints

Find disparity by subtracting xVals of matching points

Make disparity as eight times absolute value of it

Print implemented disparity map and openCV disparity map

**2) REQUIREMENTS**

Text

Description automatically generated

Python version: 3.10.4

OpenCV version: 4.5.5

**3) TESTS AND RESULTS**

Running python code with python main.py

# A picture containing graphical user interface Description automatically generated

# Running with Jupyter Notebook

# Opython files should be in the same folder with jupyter notebook file.

# Text Description automatically generated

# *Main Menu Screen to select 2 images and select implementation option*

# Text Description automatically generated with medium confidence

# There are 3 implementations of stereo correspondence to find disparity using feature-based methods.

# *Example output of “Sift With Bf\_L2”*

# A screenshot of a video game Description automatically generated with medium confidence

# A picture containing background pattern Description automatically generated

# As you can see above results that sparse disparity map looks like above. The whitest points are the points that are close to us. (Because of disparity is bigger when object is closer.)

# *Example output of “Orb With Flann”*

# Matches results:

# A screenshot of a video game Description automatically generated with medium confidence

# Ground truth and my sparse results (left is ground truth, right is my result) A picture containing text, indoor, screenshot, electronics Description automatically generated

# *Example output of “Edge with Fast Brief Norm Hamming”*

# Matches results

# A screenshot of a computer Description automatically generated with medium confidence

Ground truth and my sparse results (left is ground truth, right is my result)

Graphical user interface, application

Description automatically generated As you can see from the results above,

In the bottom of the image the disparity is more, and I set the matplotlib to ‘hot’ mode to see the results clearly. In the bottom of my results, points are denser, and their color are hotter. On the upper left side, disparity is kind of middle and in my results, there are points for it. And in the right upper, the disparity value is very low.

Matching method used is really important when we are calculating disparity. Because brute force matcher tries all possibilities to find the best match, But flann is looking for nearest neighbors, so it’s going to take less time. But results might be better with brute force because of that.

L2 norm is good choice for SIFT / SURF’s descriptors and norm hamming is good for Orb/Brisk/Brief.

As you can see above my results are not perfectly fine. The reason for that depends on these rules. Also, the keypoints found may not be compatible always. For example, if the image that we are applying edge detection is not a good ground to find the edges, canny will not find all the edges. This will be a problem when we are using keypoints and descriptors to find all the matches.

Another possible failure reason is feature set. For example, ORB is one of the fastest algorithms but SIFT gives better results for most of the cases.

In the middlebury page, ground truth disparity maps are scaled by factor 8.

So if disparity value is 120, actually this means that the pixel that corresponds to it (one of them is x, the other one of them is x’) has a difference of 120 / 8. Which is 15.

So because of that, I made my results like that too. My scale factor is also 8.

After checking every location point, I apply (x’ – x) to get the corresponding differences then I multiply by 8. I print points and disparity value for better understanding.

Graphical user interface, application, Word

Description automatically generated

My values and ground truth disparity values numerical results are like below. The values have usually a difference of 18-60. The errors are most probably because of lacks of feature sets and matching methods instead of using dense match for every point(or block.)

Text

Description automatically generated